***Module (JAVASCRIPT BASIC & DOM) - 4***

**1). What is JavaScript?**

**Ans.** JavaScript is a high-level, interpreted programming language that is primarily used for adding interactivity and dynamic behavior to web pages. It is a core technology of the World Wide Web alongside HTML and CSS, enabling client-side scripting.

Javascript is one of the most popular programming languages for web development due to its versatility and wide browser support.

Here are some key features and uses of JavaScript:

* **Client-Side Scripting**: JavaScript is primarily used for client-side scripting, meaning it runs on the user's web browser rather than the server. It allows developers to manipulate and interact with HTML elements, modify CSS styles, handle events, and validate user inputs.
* **Interactivity and Dynamic Content**: JavaScript enables the creation of interactive web experiences by responding to useractions and events. It can dynamically modify the content and behavior of a web page without requiring a page reload. This includes tasks like form validation, image sliders, interactive maps, and more.
* **DOM Manipulation**: JavaScript provides powerful APIs to interact with the Document Object Model (DOM), which representsthe structure of an HTML document. It allows developers to access, modify, and manipulate DOM elements, such as adding or removing elements, changing their styles, or updating their content.
* **Web Application Development**: JavaScript is widely used for building web applications. With frameworks and libraries like React, Angular, and Vue.js, developers can build complex and responsive web applications with ease. JavaScript also supports AJAX (Asynchronous JavaScript and XML) for making asynchronous requests to the server, enabling dynamic updates without page reloads.
* **Server-Side Development**: While JavaScript is primarily known for its client-side capabilities, it has also expanded into server-side development. With the introduction of Node.js, JavaScript can now be used as a server-side language, allowing developers to build scalable and efficient network applications.
* **APIs and Third-Party Integration**: JavaScript integrates with various APIs and services, enabling developers to interact with external resources, such as social media platforms, maps, payment gateways, and more. It facilitates communication and data exchange between web applications and external services. JavaScript has a rich ecosystem with a vast number of libraries, frameworks, and tools that enhance development productivity and enable developers to build complex web applications efficiently. It continues to evolve and remains a crucial technology for web development.

**2). What is the use of is NaN function?**

**Ans.** The **isNaN()** function is a JavaScript function used to determine whether a value is "Not-a-Number" (NaN). It takes one argument and checks if the value is a valid numeric value or can be converted into a numeric value.

The main use of the **isNaN()** function is to handle input validation and error checking when dealing with numeric data. It is often used to validate user input in forms, ensuring that the entered value is a valid number.

Here's an example of how **isNaN()** can be used:

let userInput = "abc";

if (isNaN(userInput)) {

console.log("Invalid input. Please enter a numeric value.");

} else {

console.log("Valid input.");

}

In this example, if the **user Input** is not a valid numeric value or cannot be converted into one, the **isNaN()** function will return **true**, and the code inside the **if** block will execute, displaying an error message. Otherwise, if the value is a valid number, the code inside the **else** block will execute, indicating that the input is valid.

It's important to note that the **isNaN()** function attempts to convert the argument to a number before determining its validity. So, if the argument is a string that can be successfully converted to a number, **isNaN()** will return **false**.

**3). What is negative Infinity?**

**Ans.** In JavaScript, negative infinity is a special value that represents a number that is lower than any other number.

It is denoted by the constant Number.NEGATIVE\_INFINITY.

Negative infinity is typically used to indicate a value that is beyond the lower limit of a numeric range or to represent the result of a mathematical operation that exceeds the maximum representable value.

Here are some important points to note about negative infinity:

* **Value:** Negative infinity is a numeric value that represents negative infinity. It is a special value distinct from regular negative numbers.
* **Comparison:** Negative infinity is less than any other number, including negative numbers. For example, -Infinity < -100 is true.
* **Arithmetic Operations:** When negative infinity is involved in arithmetic operations, the result is always negative infinity.

For example, (-Infinity) + 10 and (-Infinity) \* 5 both evaluate to -Infinity.

* **Range Comparison:** Negative infinity is typically used to compare against a value to check if it falls within a specific range.

For example, if you want to check if a variable x is less than or equal to negative 100, you can use x <= -100 or x > -Infinity as a comparison.

Here's an example that demonstrates the usage of negative infinity in JavaScript:

console.log(Number.NEGATIVE\_INFINITY); // Output: -Infinity

console.log(-Infinity < 0); // Output: true

console.log(-Infinity < -100); // Output: true

console.log(-Infinity + 10); // Output: -Infinity

console.log(-Infinity \* 5); // Output: -Infinity

var x = -200;

console.log(x > -Infinity); // Output: true

console.log(x <= -100); // Output: true

**4). Which company developed JavaScript?**

**Ans.** JavaScript was developed by Netscape Communications Corporation, a now-defunct software company. Brendan Eich, a Netscape employee, created the language in 1995. Initially, it was called "LiveScript" but was later renamed to JavaScript to capitalize on the popularity of Java at the time. JavaScript was primarily developed for use in Netscape Navigator, a web browser.

Netscape released the first version of JavaScript in collaboration with Sun Microsystems.

Since its creation, JavaScript has evolved and gained widespread adoption as a versatile programming language for web development.

It is now supported by all major web browsers and has become an essential part of modern web development. JavaScript is also standardized through the ECMAScript specification, which is maintained by the Ecma International standards organization.

**5). What are undeclared and undefined variables?**

**Ans.** Undeclared and undefined variables are terms used in JavaScript to describe different states of variables.

1. **Undeclared Variables:** An undeclared variable refers to a variable that has not been formally declared or defined within the scope where it is being used. If you attempt to use an undeclared variable, it will result in an error. It means that the variable has not been created or assigned a value using the **var**, **let**, or **const** keywords.

Example of using an undeclared variable:

console.log(undeclaredVariable); // ReferenceError: undeclaredVariable is not defined

In the above example, undeclared Variable is used without prior declaration, resulting in a reference error.

1. **Undefined Variables:** An undefined variable refers to a variable that has been declared but has not been assigned a value or has been explicitly assigned the value of **undefined**. When you declare a variable without assigning a value to it, it automatically takes the value of **undefined**.

Example of using an undefined variable:

let undefinedVariable;

console.log(undefinedVariable); // Output: undefined

In the above example, undefinedVariable is declared but not assigned a value. When you log its value, t will be undefined.

It's important to note that undefined is also the default value of function parameters that are not passed an argument.

function greet(name) {

console.log('Hello, ' + name);

}

greet(); // Output: Hello, undefined

In summary, an undeclared variable refers to a variable that has not been declared at all, while an undefined variable is a declared variable that has not been assigned a value or explicitly assigned the value of **undefined**.

**6). Write the code for adding new elements dynamically?**

**Ans.** To add new elements dynamically to a web page using JavaScript, you can follow these steps:

1. **Create the element:** Use the document.createElement() method to create a new element. Specify the type of element you want to create, such as <div>, <p>, <span>, etc.
2. **Set element properties (optional):** If necessary, you can set properties or attributes of the element using methods like element.setAttribute() or by directly modifying properties like element.innerHTML or element.textContent.
3. **Add content (optional):** If your element requires content, you can add it by modifying the innerHTML or textContent properties of the element.
4. **Append the element:** Choose an existing element in the document where you want to add the new element. This can be achieved by using methods like document.getElementById(), document.querySelector(), or other DOM traversal methods. Once you have the parent element, use the appendChild() or insertBefore() method to add the new element as a child of the parent element.

Here's an example that demonstrates dynamically adding a new paragraph (<p>) element to a <div> element with the id "myDiv":

// Step 1: Create the new element

const newParagraph = document.createElement('p');

// Step 2: Set element properties (optional)

newParagraph.setAttribute('class', 'highlight');

newParagraph.textContent = 'This is a dynamically added paragraph.';

// Step 3: Add content (optional)

// Step 4: Append the element

const parentElement = document.getElementById('myDiv');

parentElement.appendChild(newParagraph);

In this example, the **createElement()** method creates a new **<p>** element. The **setAttribute()** method sets the **class** attribute of the paragraph, and the **textContent** property sets the text content of the paragraph. Finally, the **appendChild()** method adds the new paragraph as a child of the element with the id "myDiv".

Make sure to adjust the code according to your specific requirements, such as the type of element you want to add and the target parent element in your HTML structure.

**7). What is the difference between ViewState and SessionState?**

**Ans.** ViewState and SessionState are both concepts used in web development to manage state information, but they serve different purposes:

1. **ViewState:** ViewState is used to store the state of a single web page in an ASP.NET web application. It is used to preserve the values of controls and other page-specific data between postbacks (when a form is submitted to the server). ViewState is encoded and sent to the client as a hidden field in the HTML, and it is automatically restored on the server when the form is submitted again.
2. **SessionState:** SessionState is used to store user-specific data across multiple web pages in an ASP.NET web application. It provides a way to persist data between different requests from the same user. SessionState data is stored on the server-side and is associated with a unique session identifier (usually stored in a cookie or URL). Each user accessing the web application gets their own unique session, allowing the application to track user-specific information.

In summary, ViewState is used to store temporary data for a single web page during postbacks, while SessionState is used to store user-specific data across multiple web pages during a user's session on the website.

**8). What is === operator?**

**Ans.** The **===** operator is a comparison operator used in programming languages such as JavaScript. It is known as the "strict equality" operator.

When you use **===** to compare two values, it checks if the values are not only equal but also of the same type. If the types of the values being compared are different, the **===** operator will return **false**.

For example, consider the following comparison:

const a = 5;

const b = "5";

console.log(a === b); // false

In this case, even though the values of **a** and **b** are the same, the types are different (a number and a string). Therefore, the **===** operator returns **false**.

On the other hand, if you use the **==** operator (loose equality) for the same comparison, it performs type coercion and considers the values equal:

console.log(a == b); // true

Here, the **==** operator converts the string **"5"** to a number and then compares the values, resulting in **true**.

In general, it is recommended to use the **===** operator for equality comparisons in JavaScript to avoid unexpected results due to type coercion.

**9). How can the style/class of an element be changed?**

**Ans.** The style and class of an element can be changed using various methods depending on the approach you prefer:

1. **Changing inline styles:**

* **JavaScript:** Use the **style** property of the element to directly modify its inline styles. For example:

// Get the element by its ID or any other selector

const element = document.getElementById('myElement');

// Change the background color

element.style.backgroundColor = 'red';

// Change the font size

element.style.fontSize = '20px';

// Change multiple styles at once using CSSText

element.style.cssText = 'background-color: blue; color: white; font-weight: bold;';

// Get the element by its ID or any other selector

const element = document.getElementById('myElement');

1. **Adding or removing classes:**

* **JavaScript:** Use the **classList** property to manipulate classes on an element. You can add or remove classes using the **add()** and **remove()** methods, respectively. For example:

// Add a class

element.className = 'newClass';

// Add a class (when multiple classes already exist)

element.className += ' anotherClass';

// Remove a class

element.className = element.className.replace('oldClass', '');

// Replace a class

element.className = element.className.replace('oldClass', 'newClass');

* **jQuery:** Use the **addClass()** and **removeClass()** methods to add or remove classes on an element, respectively. For example:

$("#myElement").addClass("newClass");

$("#myElement").removeClass("oldClass");

1. **Modifying the element's className:**

* **JavaScript:** Directly modify the **className** property of the element to replace the entire class attribute. For example:

const element = document.getElementById("myElement"); element.className = "newClass";

* **jQuery:** Use the **attr()** method to modify the value of the **class** attribute. For example:

$("#myElement").attr("class", "newClass");

These are some common methods to change the style or class of an element dynamically. Remember to select the appropriate method based on the library (JavaScript or jQuery) you are using and the specific requirements of your project.

**10). How to read and write a file using JavaScript?**

**Ans.** In a web browser environment, JavaScript alone does not have direct access to the file system for security reasons. However, you can accomplish file reading and writing using different methods and APIs provided by browsers.

1. **Reading a File (using FileReader API):**

const fileInput = document.getElementById('fileInput'); fileInput.addEventListener('change', (event) => {

const file = event.target.files[0];

const reader = new FileReader();

reader.onload = function (event) {

const contents = event.target.result; console.log(contents); }; reader.readAsText(file); });

In this example, an input element with the ID **fileInput** is used to select a file. When the user selects a file, an event listener is triggered. The FileReader API is used to read the contents of the file as text, and the result is logged to the console.

1. **Writing to a File (using Blob and download):**

const content = 'This is the content to be written to the file.';

const filename = 'example.txt';

const blob = new Blob([content], { type: 'text/plain' });

const url = URL.createObjectURL(blob);

const link = document.createElement('a');

link.href = url;

link.download = filename;

document.body.appendChild(link);

link.click();

document.body.removeChild(link);

In this example, a blob is created with the desired content and MIME type. A temporary URL is created using **URL.createObjectURL**. Then, a dynamically created anchor (**<a>**) element is used to initiate the download when clicked. After the download, the anchor element is removed from the document.

These methods allow you to read and write files in a web browser environment. Remember to handle error cases appropriately and consider any security restrictions or limitations imposed by the browser.

**11). What are all the looping structures in JavaScript?**

**Ans.** JavaScript provides several looping structures that allow you to repeatedly execute a block of code. The commonly used looping structures in JavaScript are:

1. **for loop:** The for loop is a versatile loop that executes a block of code a specified number of times. It consists of an initialization, a condition, an iterator, and a loop body.

for (initialization; condition; iterator) {

// Code to be executed in each iteration

}

Example:

for (let i = 0; i < 5; i++) {

console.log(i);

}

1. **while loop:** The while loop executes a block of code as long as a specified condition is true. It checks the condition before each iteration.

while (condition) {

// Code to be executed in each iteration

}

Example:

let i = 0;

while (i < 5) {

console.log(i);

i++;

}

1. **do...while loop:** The do...while loop is similar to the while loop, but it checks the condition after each iteration. This ensures that the loop body is executed at least once, even if the condition is initially false.

do {

// Code to be executed in each iteration

} while (condition);

Example:

let i = 0;

do {

console.log(i);

i++;

} while (i < 5);

1. **for...in loop:** The for...in loop is used to iterate over the properties of an object. It iterates over the enumerable properties of an object, including inherited properties.

for (variable in object) {

// Code to be executed in each iteration

}

Example:

const person = { name: 'John', age: 30 };

for (let key in person) {

console.log(key + ': ' + person[key]);

}

1. **for...of loop:** The for...of loop is used to iterate over iterable objects, such as arrays, strings, or collections. It provides an easy way to loop through the values of an iterable.

for (variable of iterable) {

// Code to be executed in each iteration

}

Example:

const arr = [1, 2, 3];

for (let num of arr) {

console.log(num);

}

**12). How can you convert the string of any base to an integer in JavaScript?**

**Ans.** In JavaScript, you can convert a string representing a number in any base to an integer using the **parseInt()** function. The **parseInt()** function takes two arguments: the string to be converted and the base of the number.

Here's an example of converting a string in base 2 (binary) to an integer:

const binaryString = "101010";

const decimalNumber = parseInt(binaryString, 2);

console.log(decimalNumber); // Output: 42

In this example, the string **"101010"** represents a binary number. By passing **2** as the second argument to **parseInt()**, we indicate that the string should be interpreted as a binary number. The **parseInt()** function then converts the binary string to the decimal number **42**.

Similarly, you can convert strings representing numbers in other bases such as octal (base 8) and hexadecimal (base 16) by providing the appropriate base value to the **parseInt()** function.

const octalString = "52";

const decimalNumber = parseInt(octalString, 8);

console.log(decimalNumber); // Output: 42

const hexString = "2A";

const decimalNumber = parseInt(hexString, 16);

console.log(decimalNumber); // Output: 42

In these examples, the **parseInt()** function converts the octal string **"52"** to the decimal number **42**, and the hexadecimal string **"2A"** to the decimal number **42**.

**13). What is the function of the delete operator?**

**Ans.** The delete operator in JavaScript is used to delete a property from an object or to remove an element from an array. Its function depends on the context in which it is used:

1. **Deleting Object Properties:** When used with an object, the delete operator removes a specific property from the object.

const obj = { name: 'John', age: 30 };

delete obj.age;

console.log(obj); // Output: { name: 'John' }

In the example above, the delete operator is used to remove the age property from the obj object. After the deletion, the age property is no longer present in the object.

It's important to note that the delete operator only removes own properties of an object, not inherited properties.

1. **Removing Array Elements:** When used with an array, the delete operator sets the value of the specified element to undefined, but it does not remove the element from the array. The array length remains unchanged.

const arr = [1, 2, 3];

delete arr[1];

console.log(arr); // Output: [1, undefined, 3]

In the example above, the delete operator is used to set the value of the element at index 1 to undefined.

The element is not actually removed from the array, but its value is replaced with undefined.

To remove an element from an array and update the array length, you can use methods like splice() or

create a new array without the desired element.

1. **Deleting Variables:** The delete operator cannot delete variables declared with var, let, or const. It can only delete properties of objects or elements of arrays.

var x = 10;

delete x;

console.log(x); // Output: 10 (variable is not deleted)

In the example above, the delete operator has no effect on the variable x. The variable remains intact, and its value is unchanged.

It's important to note that variables declared with let or const are block-scoped and cannot be deleted using the delete operator.

**14). What are all the types of Pop up boxes available in JavaScript?**

**Ans.** The commonly used pop-up boxes in JavaScript are:

1. **Alert:** The alert() function displays a simple pop-up dialog box with a message and an OK button. It is used to provide information or alert the user.

alert('This is an alert message.');

1. **Confirm:** The **confirm()** method displays a dialog box with a message and two buttons: OK and Cancel. It is used to ask the user for confirmation or a binary choice.

const result = confirm('Are you sure you want to proceed?');

if (result) {

// User clicked OK

} else {

// User clicked Cancel

}

The **confirm()** method returns a Boolean value (**true** if OK is clicked, and **false** if Cancel is clicked).

1. **Prompt:** The **prompt()** method displays a dialog box with a message, an input field for the user to enter a value, and OK and Cancel buttons. It is used to prompt the user for input.

const userInput = prompt("Please enter your name:");

if (userInput !== null) {

// User clicked OK and provided input

console.log("Hello, " + userInput + "!");

} else {

// User clicked Cancel or closed the prompt

console.log("No name entered.");

}

The **prompt()** method returns the value entered by the user as a string, or **null** if Cancel is clicked or the prompt is closed.

These pop-up boxes provide simple ways to interact with the user in JavaScript. They are useful for displaying messages, gathering confirmation, or obtaining user input. However, be mindful of their potential to interrupt the user experience, and consider using alternative methods for more sophisticated user interactions.

**15). What is the use of Void (0)?**

**Ans.** The use of **void(0)** in JavaScript is primarily seen as a way to prevent the browser from following a hyperlink when it is used as the **href** value of an anchor (**<a>**) tag.

By convention, when the **href** attribute of an anchor tag is set to **javascript:void(0)**, clicking on the link does not result in any navigation or page reload. It essentially acts as a placeholder or "do nothing" action. This can be useful when you want to attach event handlers to the link without triggering any navigation.

For example, you might use it in an **onclick** attribute or with JavaScript event listeners to trigger specific actions when the link is clicked, without navigating away from the current page:

htmlCopy code

<a href="javascript:void(0)" onclick="myFunction()">Click me</a>

javascriptCopy code

const link = document.querySelector('a');

link.addEventListener('click', function(event) {

event.preventDefault(); // Prevents the default link behavior

// Perform some action });

It's worth noting that the use of **void(0)** is not necessary in modern JavaScript applications or when utilizing event listeners. Instead, you can use **event.preventDefault()** or **return false** in the event handler to prevent the default action. However, in legacy code or specific scenarios, you may still encounter **void(0)** being used to prevent navigation.

**16). How can a page be forced to load another page in JavaScript?**

**Ans.** To force a page to load another page in JavaScript, you can use the window.location object to modify the URL of the current page. There are a few different ways to achieve this:

1. Setting the **window.location.href** property:

window.location.href = "https://www.example.com";

This will navigate the current page to the specified URL, loading a new page.

1. Using the **window.location.assign()** method:

window.location.assign("https://www.example.com");

This method has the same effect as setting window.location.href and is often used interchangeably.

1. Utilizing the **window.location.replace()** method:

window.location.replace("https://www.example.com");

This method also loads a new page, but it replaces the current page in the browser's history, preventing the user from navigating back to the original page using the browser's back button.

In all of these approaches, you can specify the URL of the page you want to load. It can be an absolute URL, such as "https://www.example.com", or a relative URL within the same domain, such as "another-page.html".

Keep in mind that forcing a page to load another page can result in an abrupt user experience, potentially interrupting the current task or losing unsaved data. Ensure that you provide appropriate user feedback or confirmation when performing such actions.

**17). What are the disadvantages of using innerHTML in JavaScript?**

**Ans.** While the **innerHTML** property in JavaScript is a convenient way to manipulate HTML content, it does have some potential disadvantages:

1. **Security Risks:** Using **innerHTML** to directly insert user-generated or untrusted content into your page can expose your application to security vulnerabilities like cross-site scripting (XSS) attacks. If the content is not properly sanitized or validated, malicious scripts can be injected and executed on the page.
2. **Performance Impact:** Modifying the **innerHTML** of an element causes the browser to parse and re-render the entire HTML content within that element. If you frequently update large portions of HTML using **innerHTML**, it can have a performance impact, especially on complex web pages with many elements.
3. **Event Listeners and Data Binding:** When using **innerHTML** to replace or update HTML content, any event listeners or data bindings attached to the elements within that content will be lost. You need to rebind or reattach event listeners after modifying the **innerHTML**, which can lead to additional code complexity and maintenance issues.
4. **Overwriting Existing Content:** When you set the **innerHTML** of an element, it completely replaces the existing content within that element. If there are any additional elements or data attached to the existing content, they will be lost unless you specifically handle and reattach them.
5. **Limited Error Handling:** The **innerHTML** property does not provide robust error handling or feedback if there are syntax errors or issues with the HTML content being inserted. It may fail silently or produce unexpected results if the HTML is malformed.

To mitigate these disadvantages, consider using alternative approaches like DOM manipulation methods (**createElement**, **appendChild**, etc.), template systems, or frameworks that offer more structured and controlled ways to update HTML content. Additionally, always sanitize and validate any user-generated or untrusted content before inserting it into your page.

**18). Create password field with show hide functionalities**

**Ans.**

* **HTML:**

<div>

<input type="password" id="passwordField" placeholder="Enter password">

<button id="toggleButton">Show</button>

</div>

* **CSS:**

#passwordField {

margin-bottom: 10px;

}

#toggleButton {

cursor: pointer;

}

* **Javascript:**

const passwordField = document.getElementById('passwordField');

const toggleButton = document.getElementById('toggleButton');

toggleButton.addEventListener('click', function () {

if (passwordField.type === 'password') {

passwordField.type = 'text';

toggleButton.textContent = 'Hide';

} else {

passwordField.type = 'password';

toggleButton.textContent = 'Show';

}

});

**19). Create basic math operation in JS**

**Ans.**

* **Javascript:**

// Addition

const additionResult = 5 + 3;

console.log('Addition:', additionResult); // Output: 8

// Subtraction

const subtractionResult = 10 - 4;

console.log('Subtraction:', subtractionResult); // Output: 6

// Multiplication

const multiplicationResult = 6 \* 2;

console.log('Multiplication:', multiplicationResult); // Output: 12

// Division

const divisionResult = 15 / 3;

console.log('Division:', divisionResult); // Output: 5

// Modulo (Remainder)

const moduloResult = 10 % 3;

console.log('Modulo:', moduloResult); // Output: 1

// Exponentiation

const exponentiationResult = 2 \*\* 3;

console.log('Exponentiation:', exponentiationResult); // Output: 8

**20). Create a sliders using JavaScript**

**Ans.**

* **HTML:**

<div class="slider-container">

<input type="range" id="slider" min="0" max="100" value="50">

<p id="sliderValue">50</p>

</div>

* **CSS:**

.slider-container {

width: 300px;

margin-bottom: 20px;

}

#slider {

width: 100%;

}

#sliderValue {

text-align: center;

}

* **Javascript:**

const slider = document.getElementById('slider');

const sliderValue = document.getElementById('sliderValue');

// Update the slider value display

slider.addEventListener('input', function () {

sliderValue.textContent = slider.value;

});

// Perform an action when the slider is changed

slider.addEventListener('change', function () {

const value = parseInt(slider.value);

// Perform desired action with the slider value

console.log('Slider value:', value);

});